2002-05-02

Practical and efficient construction of network caricatures

Version	
Citation (published version):	A Bestavros. 2002. "Practical and Efficient Construction of Network Caricatures."

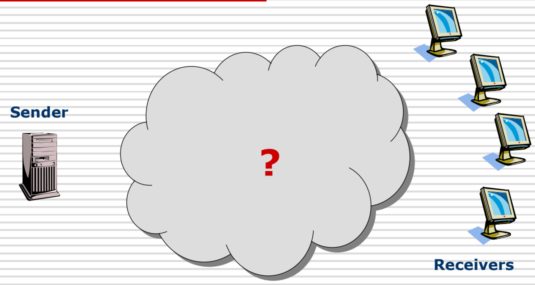
Practical and Efficient Construction of Network Caricatures

Azer Bestavros

Joint work with
Khaled Harfoush & **John Byers**

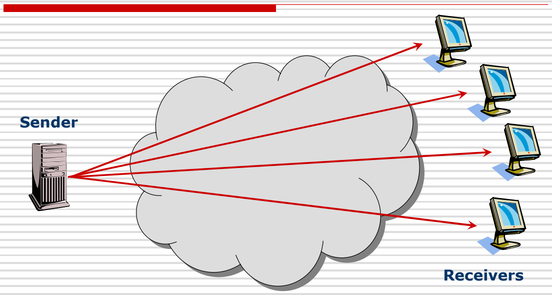
Web and interNetworking Group
http://www.cs.bu.edu/groups/wing

Q: What does the net look like?



Using only end-to-end observations

A: A bunch of "independent" pipes

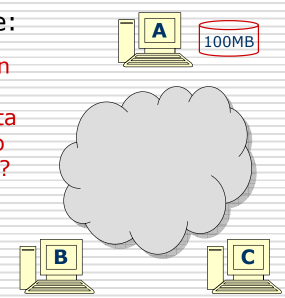


That's how TCP views it

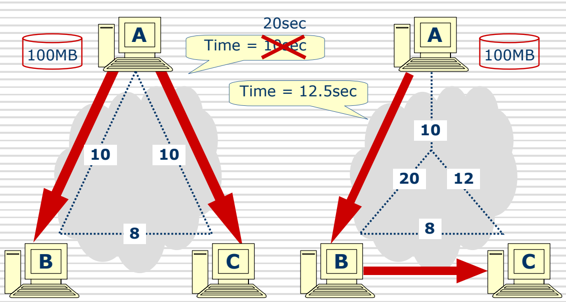
Q: Why is that not good enough?

A motivating example:

- How to construct an overlay network to move 100MB of data from grid node A to grid nodes B and C?



A: The devil is in the details!



Other motivating examples…

Aggregate congestion control:

- How to partition a set of flows into "congestion-equivalent" classes?

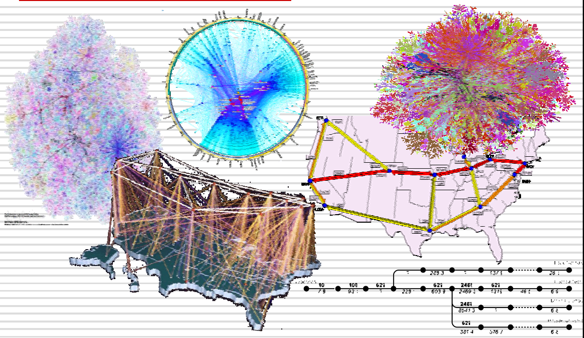
Parallel downloads from multiple servers:

- Which m (out of n) servers to select to maximize aggregate download bandwidth?

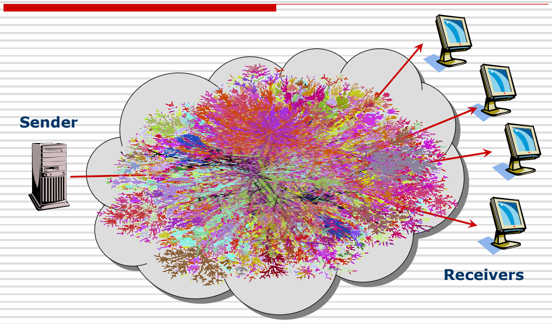
Request scheduling at busy media servers:

- How to prioritize requests to avoid competition for shared network resources?

Q: Why not use physical net maps?



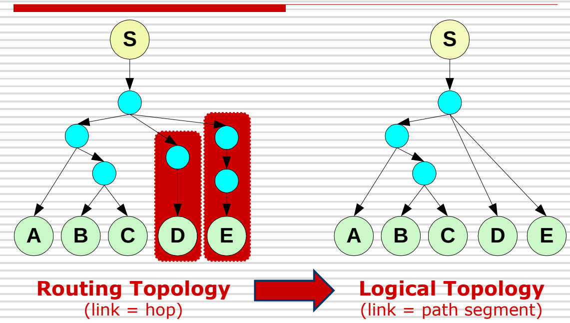
A: It's overkill!



Q: How much detail is enough?

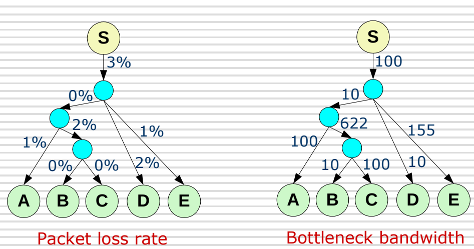
- Need to determine the **level of sharing** between a sender and any subset of receivers **w.r.t. a metric** of interest.
- Example metrics:
  - *Number of hops (e.g., router or AS hops)*
  - *Delay (e.g., transmission or queuing delays)*
  - *Bandwidth (e.g., available or bottleneck)*
  - *Packet loss rate*

From routing to logical topologies



Labeled logical topologies

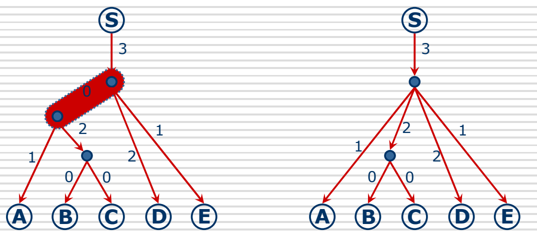
- Labeling a logical topology quantifies the sharing among subsets of endpoints (w.r.t. a metric)



Hiding uninteresting structures

- Uninteresting links in the logical topology are those with metrical values above or below certain thresholds of interest
- Examples:
  - Hide links with losses below 2%
  - Hide links with bandwidth above 100 Mbps
  - Hide links with delays below 1msec

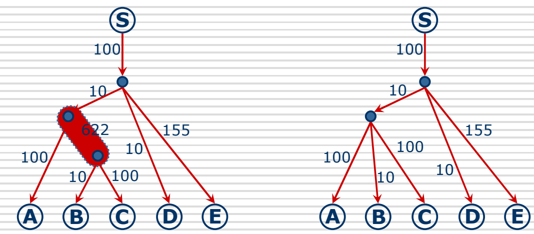
Hiding unobservable segments



Logical Topology (labels = loss rate)

Loss-Induced Topology (labels = loss rate)

Hiding over-provisioned segments

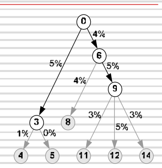


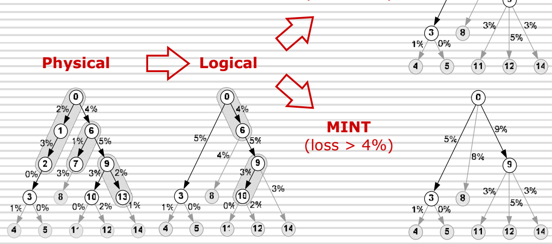
Logical Topology (labels = bandwidth)

BW-Induced Topology (labels = bandwidth)

Metric-Induced Network Topology

MINT = a summary of the level of sharing between a sender and any subset of receivers w.r.t. a metric of interest





Problem statement

- Given:
  - A set of senders and receivers
  - A metric and associated sensitivity threshold
- Do the following:
  - **Infer** the MINT observable at a sender
  - **Label** the inferred MINT
  - **Integrate** MINTs obtained at different times
  - **Integrate** MINTs obtained at different senders

Two approaches:

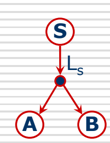
- Prior Work:

  Solve the problem using "metric-specific" or "technology-specific" techniques, e.g.:
  - Determine if paths share a bottleneck [RKT'00]
  - Quantify shared losses [CDLPT'99] [HBB'00]
  - Quantify shared bottleneck bandwidth [HBB'01]
- Our Work:

  Provide generic solutions that work on a variety of metrics satisfying certain properties

MINT Framework [BBH:Infocom02]

- Assumes the availability of an "oracle" that quantifies the sharing $f(L_S)$ between a sender and any two receivers A and B.



- Depending on the properties of the metric of interest, provides sound constructions that enable MINT inference, labeling, and integration.

Metric properties

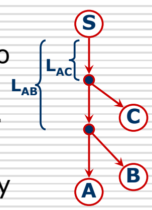
- **Monotonicity:**
  $f(L_{ij}) \le f(L_{ik})$ or
  $f(L_{ij}) \ge f(L_{ik})$
- **Separability:**
  $f(L_{jk}) = g\big(f(L_{ik}), f(L_{ij})\big)$
- **Symmetry:**
  $f(L_{ij}) = f(L_{ji})$

Metric Properties: Examples

	Monotonic	Separable	Symmetric
Loss Rate	✓	✓	
Queuing Delay	✓	✓	
1-way Prop Delay	✓	✓	?
2-way Prop Delay			✓
Hop Count	✓	✓	?
Bottleneck Bandwidth	✓		?
Jitter			

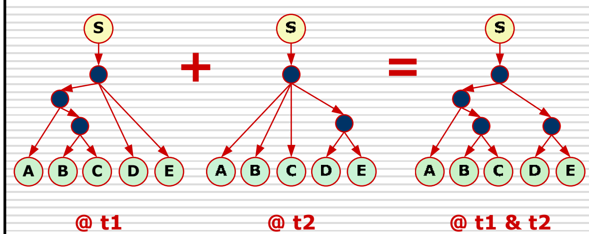
From properties to constructions

- If $f(L_{AB}) > f(L_{AC})$ then monotonicity implies ability to order A, B, C in terms of where they join the topology.
- Theorem: Metric monotonicity enables inference of partially (prefix) labeled MINTs.



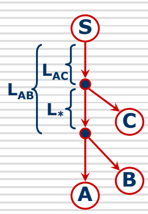
From properties to constructions

- Theorem: Metric monotonicity enables integration of MINTs observed over time.



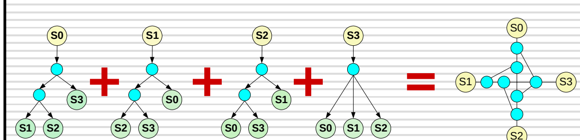
From properties to constructions

- If f() is separable, then given L_{AB} and L_{AC}, we can infer L_*.
- Theorem: Metric separability enables complete labeling of MINTs.

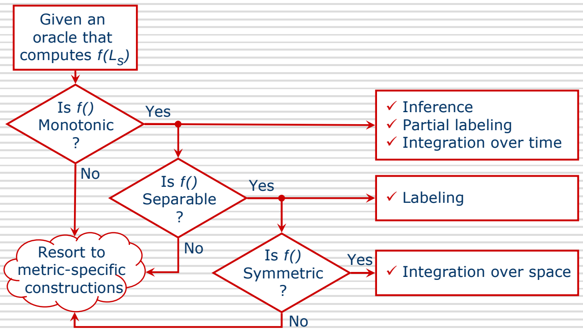


From properties to constructions

- Theorem: Metric symmetry enables the merger of MINTs observed from multiple vantage points



MINT Framework: Constructions



How about that "Oracle"?

- The oracle is simply a procedure that enables a sender to establish the level of sharing between any pair of receivers.
- Many MINT instantiations; each one is associated with a metric and an oracle.
- A MINT instantiation is as good as the oracle it relies on for that instantiation!

MINT example: Hop topologies

Metric: Hop count (h) is monotonic and separable, but not necessarily symmetric.

Oracle: Given receivers A and B, find number of hops shared between paths S→A and S→B.

MINT: Using our constructions, we can efficiently (better than linear in diameter of network) infer and label the hop topology between a sender and n receivers.

MINT example: Loss Topologies

Metric: Packet loss probability (p) is monotonic and separable, but not symmetric.

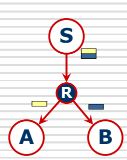
Oracle: Given receivers A and B, find p for the shared segment of paths S→A and S→B. Many methods available.† Take your pick!

MINT: Using our constructions, we can infer and label loss topologies between a sender and n receivers.

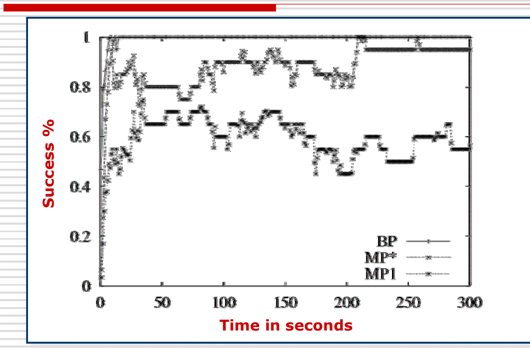
† Multicast [CDLPT'99], Poisson [RKT'00], and Bayesian probing [HBB'00]

Loss Oracle

- Many to chose from!
- Bayesian Probing [HBB:icnp'00]
  - Send two packets back-to-back, each to a different client
  - Measure probability of different outcomes
  - Relate outcome probabilities to analytical prediction (using a simple queuing model) to estimate shared loss rate
  - Has good accuracy and convergence characteristics

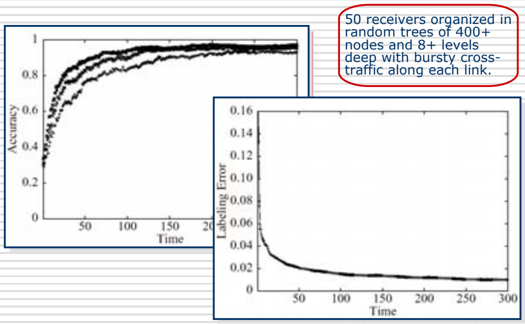


Loss Oracle: Evaluation



Loss Topology: ns Experiments

50 receivers organized in random trees of 400+ nodes and 8+ levels deep with bursty cross-traffic along each link.



MINT example: Bottleneck B/W

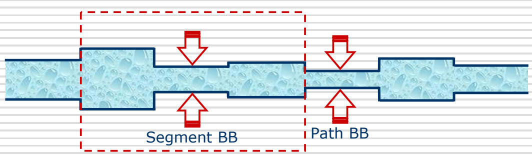
Metric: Bottleneck Bandwidth (BB) is monotonic, but neither separable nor symmetric.

Oracle: Given receivers A and B, find bb for the shared segment of paths S→A and S→B using Cartouche probing [HBB:Infocom'03].

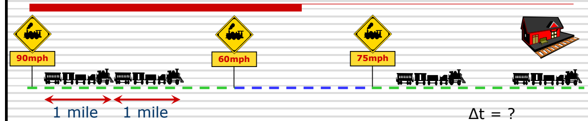
MINT: Using our constructions, we can infer and partially label the BB topology between a sender and n receivers. To completely label that topology, we need BB-specific techniques (because bb is not separable) [HBB:Infocom'03].

Bottleneck Bandwidth Estimation

- BB is the speed (capacity) of the slowest physical link along a sequence of links
- Existing path BB estimation techniques:
  - End-to-end BB [K'91][P'96][CC'96][DRM'01]
  - Hop-by-hop BB [D'99][LB'01]



Separation as a Measure of speed

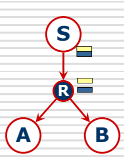


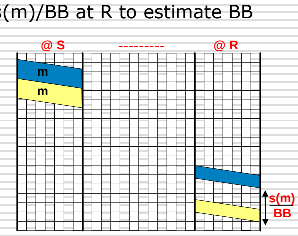
How far behind will the 2nd train be when the 1st arrives to station?

$$\text{Delay} = \frac{\text{Length of train}}{\text{fn(speed of slowest railroad segment)}}$$

Leveraging the Packet Pair Technique

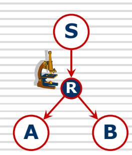
- How could we estimate the BB of the shared segment S→R ?
  - Send a packet-pair [mm] from S
  - Use separation Δ = s(m)/BB at R to estimate BB





Towards a BB Oracle

- … but how can we measure the separation at R from endpoints?
  - Need to "preserve" separation so we may measure it at an endpoint



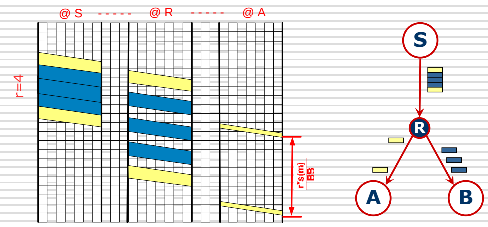
Lemma:

Separation Δ is preserved if BB(R→A) > s(m)/Δ

- s(m) is the size of the probing packets
  → Cannot make s(m) arbitrarily small
- Δ is a function of BB(S→R)
  → Need a function that yields a large Δ

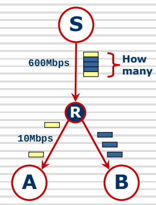
Magnifying Δ: Packet Pair Trains

- Use overlapped packet pairs to make Δ large enough at R. All but first and last packets are diverted (or dropped) at R.



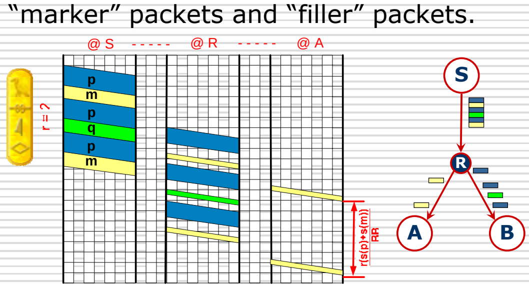
Magnifying Δ: Packet Pair Trains

- If BB(S→R) = 64 * BB(R→A), we would need to send 65 back-to-back packets to ensure a large enough Δ at R.
- Not practical!



Cartouche Probing [HBB'01]

- Recognizes difference in function between "marker" packets and "filler" packets.



Cartouche Probing [HBB'01]

- Filler packets are used to "produce" Δ
  ➔ larger is better
- Marker packets are used to "measure" Δ
  ➔ smaller is better
- Make marker packets as small as possible and filler packets as large as possible
  s(m) = 40 bytes
  s(p) = 1,500 bytes

Cartouche Probing [HBB'01]

- From S, send a probe **[pm {pq}$^{r-1}$ pm]**
  - "Filler" packets (p & q) go to B
  - "Marker" packets (m) go to A
  - s(p)>> s(m) = s(q)
- At A, measure the separation Δ between markers and calculate the shared BB:

$$BB = \frac{r\,[s(p)+s(m)]}{\Delta}$$

Cartouche Size

- Recall that to survive trip from R to A, the separation Δ must be > s(m)/BB(R→A)

  r[s(p)+s(m)]/BB(S→R) > s(m)/BB(R→A)

$$r > \frac{1}{k}\,\frac{BB(S\rightarrow R)}{BB(R\rightarrow A)}$$

- For s(m)=40 and s(p)=1500 ➔ k=38.5
- If BB(S→R) = 50 * BB(R→A), we would need to send a cartouche of size r=2, or a total of 6 packets

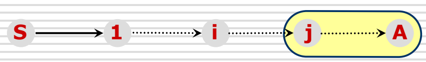
Path BB estimation: Summary

	Probe Structure		
	Packet Pair [mm]	Packet Pair train [m{m}$^{r-1}$m]	Cartouche [pm{pq}$^{r-1}$pm]
Separation	s(m)/BB	r s(m)/BB	r[s(m)+s(p)]/BB
Tolerable BB(S→R):BB(R→A)	1	r	38.5 r

Labeling BB MINTs

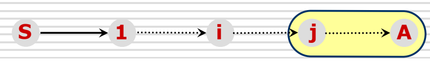
- MINT + Cartouche Probing allow us to:
  - Infer BB topology between a set of endpoints
  - Partial (i.e. prefix) labeling of BB topology
- Need BB-specific techniques to label non-prefix subpaths
- Conjecture: If we can measure the BB of a path suffix, we can in effect measure the BB of any path segment!

Path Suffix BB Estimation



- Use packet pairs to measure BB(S→A)
- Use Cartouche to measure BB(S→j)
- If BB(S→A) < BB(S→j)
  Then BB(j→A) = BB(S→A)
  Else BB(j→A) > BB(S→j)

Path Suffix BB Estimation

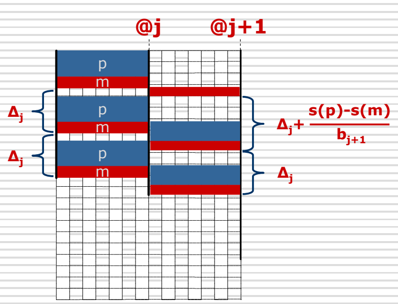


Consider a Cartouche train of the form:

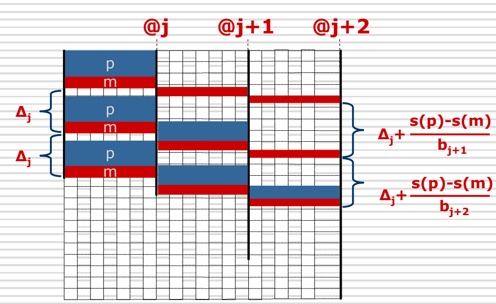
pm{pq}$^{r-1}$pm{pq}$^{r-1}$pm{pq}$^{r-1}$pm…

drops out at j — drops out at j+1 — drops out at j+2 …

Cartouche Train: Illustration (r=1)



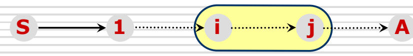
Cartouche Train: Illustration (r=1)



Path Suffix BB Estimation

- Lemma: The largest Δ between markers of the Cartouche train corresponds to the suffix BB link
- Corollary: If BB(R→A) > BB(S→A) then using a Cartouche train, we can estimate BB(R→A)

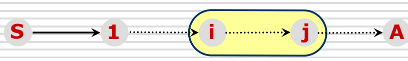
Subpath BB Estimation



- We need to preserve the spacing Δ of markers of a Cartouche train at j
- Recall that preserving Δ is a matter of sizing the Cartouche probes; i.e.

$$r > \frac{1}{k} \frac{BB(S\rightarrow j)}{BB(j\rightarrow A)}$$

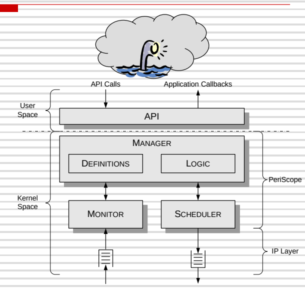
Subpath BB Estimation



- Theorem: Using an appropriately size Cartouche train, we can measure the bottleneck bandwidth of an arbitrary segment i→j on a given path S→A
- Corollary: Using Cartouche probes we can infer and label BB MINTs

PeriScope†: Linux API [HBB:Pam'02]

- A kernel-level API for implementing various probing structures
- A user-level library that implements the MINT constructions (i.e., inference, labeling, and integration)
- Used effectively to implement many MINT instantiations



† A Probing Engine for the Recovery of Internet Subgraphs (Available from WING web pages at http://www.cs.bu.edu/groups/wing)

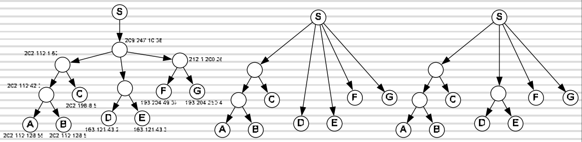
Probing Structures

- Example probing structures:
  - [pp]: Packet-pair probing [CC96] [P96] [P97].
  - [pp]: Bayesian probing [HBB00].
  - [pq]: Tailgated-pair probing [LB00].
  - $[pm(pq)^{r-1}pm]$: Cartouche Probing [HBB01].
- Probing structures differ in:
  - The number of packets.
  - The size of each probe packets.
  - The destination of each probe packet.
  - The inference function.

PeriScope Design Rationale

- Ensure kernel code modularity and restrict changes to the networking stack.
- Minimize user/kernel boundary crossings.
- Provide enough primitives to enable the definition of arbitrary probing structures and techniques.
- Provide a structured and well-defined interface for applications.

Loss Topology: Internet tests



Logical topology of hand-picked set of receivers

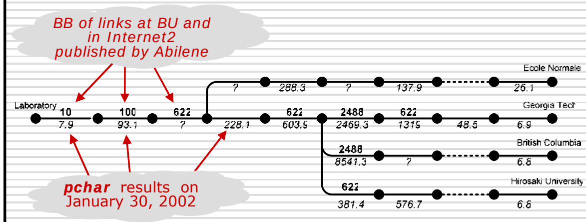
Loss topology inferred most of the time by PeriScope

Integration over time of inferred loss topologies

Using BP loss oracle implemented in PeriScope

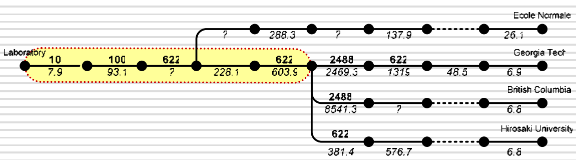
Cartouche Probing: Internet Tests

- Implemented in PeriScope
- Tested against **pchar** on path prefixes of known link speeds



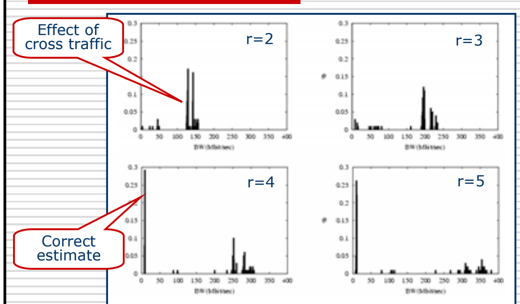
Cartouche Probing: Internet Tests

- Shared BB Experiments



- Example: Measure BB shared between the BU→GTech and BU→UBC paths

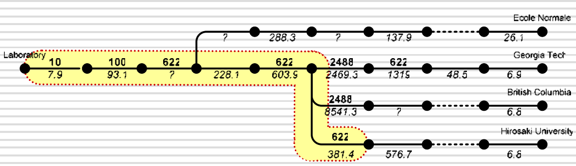
Cartouche Probing: Internet Tests



Shared BB for GTech and UBC

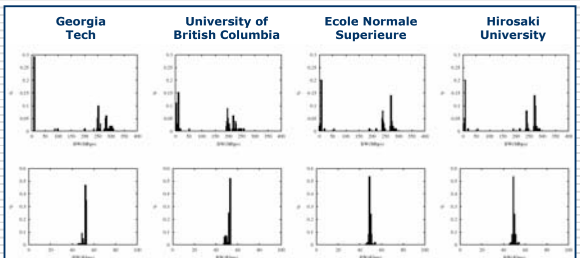
Cartouche Probing: Internet Tests

- Path Prefix BB Experiments



- Example: Measure BB of first 6 hops from BU to all destinations

Cartouche Probing: Internet Tests



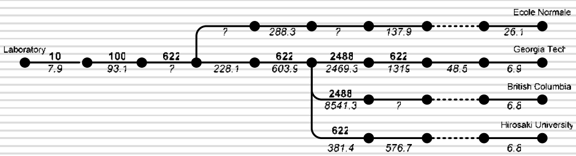
$b_{1,5}$ over a 10Mbps LAN (top) and over 56Kbps modem (bottom) using Cartouches of size r=4

Cartouche Probing: Internet Tests

Estimates of BB of arbitrary path segments

95 percentile confidence interval estimates with r=4

	b_1	$b_{1,3}$	$b_{1,6}$	b_4	$b_{4,6}$
www.gatech.edu	8 [5.22,8.77]	9 [6.15,9.49]	10 [8.99,10.54]	302 [299.81,305.18]	310 [307.70,314.80]
www.ubc.ca	7 [4.87,7.85]	7 [5.43,8.34]	8 [6.18,9.03]	286 [284.97,287.19]	178 [175.91,180.79]
www.hirosaki-u.ac.jp	7 [5.04,7.11]	7 [5.06,6.81]	7 [6.71,7.73]	185 [180.46,186.45]	184 [179.75,184.04]
www.ens.fr	8 [7.75,9.96]	8 [5.93,8.64]	7 [4.96,7.84]	Packet Reordering	Packet Reordering



Summary

- MINT is an effective framework for the representation of the sharing structure between a set of endpoints.
- We embodied MINT in a Linux API (PeriScope), which we for inference and labeling of loss, BB and delay topologies.

Food for thought

- How do we “better” validate all this?
- Active versus passive?
- Other metrics? Other properties?
- MINT “triangulation”? MINT “beacons”?
- MINT-enabled applications?
- MINT for non Internet networks?
- Caricatures for non-internet networks…

http://www.cs.bu.edu/groups/wing